RTSAT — An Optimal and Efficient Approach to the Task Allocation Problem in Distributed Architectures*

Alexander Metzner
OFFIS

Christian Herde
Carl-von-Ossietzky Universität Oldenburg

Abstract

We present an advanced SAT-based approach to the task and message allocation problem of distributed real-time systems. In contrast to the heuristic approaches usually applied to this problem, our approach is guaranteed to find an optimal allocation for realistic task systems running on complex target architectures. Our method is based on the transformation of such scheduling problems into nonlinear integer optimization problems. The core of the numerical optimization procedure we use to discharge those problems is a solver for arbitrary boolean combinations of integer constraints. While the determination of the task and message placement is done within the satisfiability checking based solver, checking for feasibility w.r.t real-time requirements is performed in a specialized real-time engine under control of the satisfiability solver. Optimal solutions are obtained by imposing a binary search scheme on top of that solver. Experiments show the applicability of our approach to industrial-size task systems.

1 Introduction

Electronic system development for automotive applications is currently undergoing major changes to cope with the exponential growth of functionality offered by cars.

First, the old paradigm of equating *one* function to *one* electronic control unit (ECU) delivered by *one* supplier is broken: new functions "tap" information on the car status from multiple sources, and rely on the proper interplay of e.g. power-train and break systems in advanced stability protection applications. Hence the implementation of such car functions involves distributed tasks sets running on multiple ECUs, with bus-based inter-task communication. This induces several challenges:

1. Application derived end-to-end latencies must now be established across shared bus systems, entailing the need to perform cross ECU network schedulability analysis incorporating models of the underlying communication architecture. This drastically raises the complexity of schedulability analysis, due to the multiple and interrelated dimensions of allocation decisions, induced communication costs, and induced processor load, rendering previous approaches ineffective.

2. Such analysis must meet the requirements of a multi-organizational design process, in which multiple suppliers develop subsets of such a system, with system- and integration responsibility resting with the OEM.[1] This entails, that OEMs must be able to assess the overall design space for possible target architectures as well as the feasibility for cost efficient realizations of new functions prior to subcontracting suppliers for pieces of the design. It will also typically require, for cost-reasons, to tighten initial solutions found during early design space exploration, taking into account refined knowledge of detailed WCET and communication deadline characteristics once suppliers provide subsystems ready for integration. The more refined analysis required for finding optimal solutions comes with yet another level of complexity, hence further making the case for highest efficiency of algorithmic methods.

Testing feasibility alone is already NP-hard[1]. In fact resource conditions alone, like memory constraints, make the mapping problem as difficult as the well-known *bin packing problem*. Thus to tackle these challenges, one has to solve a difficult combinatorial optimization problem. We do this with an algorithm which is based on use of satisfiability checking, which is a well known technique in the area of formal verification and which we extend to the real-time problem domain.

*This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS)

[1] Original Equipment Manufacturers, such as BMW, DaimlerChrylser, GM, etc. In practice, also top-level tier one suppliers such as Bosch take over integration responsibility, and hence act from process considerations in both the OEM and the supplier role.

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

We provide an efficient feasibility test of distributed tasks systems and compute solutions which are optimal w.r.t. a variety of objective functions while guaranteeing (in a rigid, mathematical sense) task deadlines and communication deadlines of the underlying tasks systems. Some exemplary objectives are to minimize the number of necessary processors, near equal utilization of processors, minimize communication over the bus-system, maximize the slackness of the system and many more.

In this paper we propose a model and algorithms to implement these ingredients for the mapping problem. Thereby we obtain an efficient and optimal automatic approach to solve the mapping problem.

Temporal predictability (as a key property) entails both the scheduling of tasks on ECUs as well as the transmission of messages over communication networks. Predictable real-time behavior of task systems can be achieved through scheduling analysis, which has been investigated intensively in the past. [2] gives a survey on this topic. Accurate time prediction for the communication of messages on lifelike bus systems was shown akin to scheduling analysis of tasks in [3]. We will use these results which provide a rich task model that enables the application of our method to real-world problems.

In the area of task allocation there are several approaches of mostly heuristic type for finding a mapping of tasks to ECUs and messages to networks. To simplify matters, several approaches restrict the architecture to a fixed specified interconnection of ECUs (c.f. [4]). Weaker architectural restrictions are taken in [5], where a set of uniform ECUs is connected by a realistic bus system. Based on the results of [3], [5] attacks the allocation problem by a simulated annealing approach. Treatment of certain additional placement restrictions, like forbidden placements and memory consumption, have been incorporated. The authors of [6] propose a combined branch & bound and list-scheduling approach to allocate tasks and messages of time-triggered and event-triggered task sets to ECUs with a time-triggered, TDMA-based bus system. Again, only very simple topologies are supported. In the area of synthesis for HW/SW codesign, some approaches include the task allocation problem, but usually use only very rough prediction of run-times and message latencies. As an example, the reader is referred to [7], where evolutionary algorithms perform an allocation of tasks to hardware or ECUs, thereby using a static schedule without preemption.

In contrast to such incomplete heuristic approaches which do not guarantee to find an optimal solution we propose an optimal strategy to assign tasks and messages to ECUs and network busses in complex architectures. We have done this by modeling timing and resource restrictions as a set of integer formulae [8]. Adding a cost function reflecting the run-time overheads of a certain task allocation, we are able to determine the assignment of tasks with minimal cost. Optimal assignments are generated by a sequence of calls to an appropriate propositional SAT checker[9], after transformation of the integer formulae into propositional formulae over the booleans. However, the approach presented in [8] suffers from large run-times and did not scale well for more complex scheduling analysis, thus in this paper we replace the solver used in [8] by a new type of SAT solver which combines experiences from hybrid system verification and scheduling analysis.

Optimal approaches for the task allocation problem were already proposed in [10] and [11]. In [10], mixed integer linear programming is used to synthesize a static schedule of tasks on an ECU in a HW/SW co-synthesis environment. Due to the restrictions of the co-synthesis domain, the task model is quite simple: all tasks have the same period and are non-preemptive and, therefore, do not suffice task models we aim at. Furthermore communication overheads are not modeled exactly and are used only for simple direct links between ECUs. [11] presents an optimal branch and bound algorithm for the task allocation problem on a richer task model that allows non-uniform periods for tasks, but is restricted to non-preemptive scheduling, too. Again, the communication model is approximate.

Contrary to the last two approaches, we aim at fixed priority-driven preemptive schedules for tasks. Furthermore, our approach provides exact analysis of message delivery cost on different bus systems, like bus systems driven by time division multiple access (TDMA) or event-driven bus systems, like CAN. Additionally, we consider cost for messages arising while crossing gateway nodes.

The paper is organized in 6 sections. Section 2 describes the basic model, section 3 illustrates the transformation of the task allocation problem into a set of integer-arithmetic formulae and presents our optimization technique. In section 4 we describe the application specific extension of the satisfiability solver dealing with real-time requirements. Section 5 presents experimental results and shows the usage of our methods in a broad range of application scenarios. Finally, section 6 concludes the paper.

2 Basic model

In order to formalize the system behavior of a real-time system within the time domain, an abstract model has to be created which defines on one hand the system architecture and on the other hand a task model which describes the application running on that architecture, plus the timing constraints the application must fulfill. A system architecture consists of a number of ECUs and a number of communication media the ECUs are connected to. In order to show the applicability of our approach we additionally introduce memory consumption as representative

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

for other resources. Thus the architecture is described by a tuple $\mathcal{A} = (P, \mu^{\mathcal{A}}, K)$, where P is the set of ECUs, $\mu^{\mathcal{A}} : P \to \mathbb{N}$ is the amount of memory attached to each ECU and $K \subseteq 2^P$ is the set of communication media.
In order to perform the task allocation and the scheduling analysis some assumption about the underlying scheduling strategy must be made. In this paper tasks are scheduled by a preemptive, fixed priority algorithm[5].

A task may send messages at the end of each computation to one or more other tasks. The arrival of a message on an ECU may activate the receiving task. The timing constraints exist for each task and each message. The task model is defined by a set $\mathcal{T}$ of tuples $(t_i, c_i, \mu_i^{\mathcal{T}}, \gamma_i, \pi_i, \delta_i, d_i)$ describing the individual tasks. The elements are the activation period or minimal inter-arrival time $t_i \in \mathbb{N}$, the worst case execution times (WCET) $c_i : P \to \mathbb{N}$, memory consumption of the task on each ECU $\mu_i^{\mathcal{T}} : P \to \mathbb{N}$, the messages (including their target, size and their deadline) the task is sending $\gamma_i \subseteq \mathcal{T} \times \mathbb{N} \times \mathbb{N}$, and the ECUs the task is allowed to be allocated on $\pi_i \subseteq P$. Tasks from δ_i are not allowed to be allocated together with τ_i (set of redundant tasks), and $d_i \in \mathbb{N}$ is the deadline of τ_i[2]. The priority $\xi(\tau_i)$ of a task τ_i is determined using the deadline monotonic approach.
Task allocation is now defined by the mapping $\Pi : \mathcal{T} \to P$ that assigns each task in $\mathcal{T}$ to an ECU.
Given a certain allocation, scheduling analysis evaluates whether all tasks can meet their timing constraints. For each task τ_i its worst case response time r_i is calculated. The same is done for message transmissions. In the remainder of this section it is outlined how the analysis is defined.

The scheduling analysis for priority ordered execution in its simplest form can be expressed by the following fixed point equation (cf. [2]):

$$r_i^{n+1} = c_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i^n}{t_j} \right\rceil c_j, \quad r_i \leq d_i \tag{1}$$

where $hp(i)$ is the set of tasks running on the same ECU with higher priority. The iteration either ends with some n when $r_i^{n+1} = r_i^n$ or when r_i^n exceeds the given deadline.

An appropriate way to model message transmission is to exploit the analogy between arbitration of a bus by messages and CPU arbitration of an ECU by tasks. A comprehensive outline of this mechanism is, e.g., given in [3] for priority driven busses as well as for TDMA based busses.

Based on the calculated response times for tasks and messages, the feasibility of the schedule can be provided by comparing them with their deadlines.

Using the same encoding scheme, many more temporal properties like release jitter, blocking factors, etc., can be represented. While this is done in our actual model we refrain from expanding on this due to lack of space.

[2]For simplicity reasons we assume that $d_i \leq t_i$. In fact, out method is able to treat even more complex systems.

3 Task allocation using satisfiability checking

3.1 Transformation of the allocation problem

In order to find (and optimize) a valid assignment of tasks to ECUs, we describe the effect of a mapping on the response times with a formula which is a conjunction of a set of arithmetic formulae over integers. In the following, all variables of the formulae are denoted by using a `typewriter` font. All other identifiers are constants that can be calculated at transformation time.

Let $\mathtt{a}_i^p$ be the boolean allocation variable for task τ_i on ECU p which assigns τ_i to ECU p. Clearly

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \sum_{\forall p \in P} \mathtt{a}_i^p = 1, \tag{2}$$

as each task is only assigned to one ECU. Let $\mathtt{a}_i$ be an integer variable representing the number of ECU p task τ_i is assigned to. Hence $\mathtt{a}_i$ represents $\Pi(\tau_i)$. The optimization procedure can assign values according to the valuation of $\mathtt{a}_i^1, \ldots, \mathtt{a}_i^p$ to $\mathtt{a}_i$, but it has to consider the placement constraints π_i and the set of redundant tasks from δ_i. We formulate this by the following in-equations:

$$\left(\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P \setminus \pi_i} \mathtt{a}_i \neq p \right) \wedge \left(\bigwedge_{\forall \tau_i \in \mathcal{T} : \delta_i \neq \emptyset} \bigwedge_{\forall \tau_j \in \delta_i} \mathtt{a}_i \neq \mathtt{a}_j \right) \tag{3}$$

The allocation of tasks to ECUs reduces the available memory resources of this ECU. Thus we have to check for each assignment whether an allocation on an ECU needs more memory than available. This can be enforced by

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall p \in P} \begin{array}{l} \big((\mathtt{a}_i = p) \rightarrow \mathtt{m}_p^i = \mu_i^{\mathcal{T}}(p)\big) \\ \wedge \big((\mathtt{a}_i \neq p) \rightarrow \mathtt{m}_p^i = 0\big) \end{array} \tag{4}$$

$$\bigwedge_{\forall p \in P} \mu^{\mathcal{A}}(p) - \sum_{\forall \tau_i \in \mathcal{T}} \mathtt{m}_p^i \geq 0 \tag{5}$$

where $\mathtt{m}_p^i$ is the amount of memory consumed by task τ_i on ECU p. In order to guarantee real-time constraints given by deadlines, we have to transform the response time analysis from equation (1) into a set of in-equations.

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \mathtt{r}_i = c_i + \sum_{\forall \tau_j \in \mathcal{T}} \mathtt{pc}_i^j, \tag{6}$$

where $\mathtt{pc}_i^j$ describes the preemption costs induced by higher priority tasks. For each task τ_j that is assigned to the ECU $\Pi(\tau_i)$ and that has a higher priority than τ_i

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

($\mathtt{p}_i^j = 1$), the WCET multiplied by the number of preemptions (caused by this task) determines the interference cost. This can be expressed with the following set of equalities:

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \big(\mathtt{p}_i^j \wedge (\mathtt{a}_i = \mathtt{a}_j)\big) \rightarrow \mathtt{pc}_i^j = \mathtt{I}_i^j \cdot c_j \qquad (7)$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \neg\big(\mathtt{p}_i^j \wedge (\mathtt{a}_i = \mathtt{a}_j)\big) \rightarrow \mathtt{pc}_i^j = 0 \qquad (8)$$

where $\mathtt{I}_i^j$ describes the number of preemptions of a task τ_i by a task τ_j. This number depends on the priority $\mathtt{p}_i^j$ of the pair of tasks. With the given priority and the given mapping $\Pi(\tau_i)$ we can now solve the recurrence equation (1), which is modeled in equation (6) using the number of preemptions $\mathtt{I}_i^j$ from each task in equation (7) and (8). $\mathtt{I}_i^j$ is a substitution of the ceiling function in (1) and we can derive the upper and lower bound of $\mathtt{I}_i^j$:

$$\left\lceil \frac{r_i}{t_j} \right\rceil =: \mathtt{I}_i^j \overset{def.\lceil\rceil}{\Longrightarrow} \frac{r_i}{t_j} \leq \mathtt{I}_i^j < \frac{r_i}{t_j} + 1$$

Because $\mathtt{I}_i^j$ is an integer variable and taking the definition of the ceiling function into account, each assignment of $\mathtt{I}_i^j$, that satisfies above conditions, equals the result of the term $\left\lceil \frac{r_i}{t_j} \right\rceil$. Furthermore, the structure of these conditions forces valid assignments to be a solution of the recurrence equation. Now these conditions can be translated into a set of inequalities for each task, where we have to ensure that tasks which are assigned to different ECUs don't preempt each other. Avoidance of interference due to lower priorities is enforced in equation (8).

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} \begin{array}{l} (\mathtt{a}_i = \mathtt{a}_j) \rightarrow ((\mathtt{I}_i^j \cdot t_j \geq \mathtt{r}_i) \\ \qquad \wedge ((\mathtt{I}_i^j - 1) \cdot t_j < \mathtt{r}_i)) \end{array} \qquad (9)$$

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \bigwedge_{\forall \tau_j \in \mathcal{T} \setminus \{\tau_i\}} (\mathtt{a}_i \neq \mathtt{a}_j) \rightarrow \mathtt{I}_i^j = 0 \qquad (10)$$

Finally, for each task we have to check whether the response time is less than or equal to the deadline of this task

$$\bigwedge_{\forall \tau_i \in \mathcal{T}} \mathtt{r}_i \leq \mathtt{d}_i \qquad (11)$$

The effect of the transmission of a message $m_{i,j}$ sent from task τ_i and received by task τ_j is modeled in a similar way.

Other requirements can be incorporated in a similar way. In the following we will refrain from describing this in detail. For more details, see [8].

3.2 A SAT–based approach to integer optimization

The encoding sketched in the previous section reduces the problem of finding an optimal allocation of tasks to an optimization problem over the integers. The constraints of this optimization problem are given as a boolean combination ϕ of linear and non-linear integer equations and in-equations, while optimality amounts to minimizing the value of some integer variable i occurring in ϕ subject to these constraints. This does in principle constitute a standard integer optimization problem, yet the formulae ϕ obtained from above encoding feature a peculiar structure which calls for extremely powerful optimization procedures: they are massively disjunctive, representing the plethora of choices entailed in the scheduling problem, and besides large linear (in)-equation systems over the integers, they do also contain a non-trivial number of non-linear integer constraints (for time triggered media, see [8]).

These formulae are thus hardly amenable to conventional numeric optimization procedures, which forced us to employ a reduction to propositional satisfiability solving (SAT) which is possible due to the bounded range of all integer variables entailed. Therefore we use the constraint solver GOBLIN [9] for this tasks. GOBLIN is able to deal with integer arithmetic within the equations by transforming all integer variables into their 2's complement encoding and using principles from the circuit synthesis to encode arithmetic operation. Thus, GOBLIN is well suited for our problem. Using this decision procedure, satisfiability of the constraint system ϕ can be decided and, if applicable, a satisfying assignment σ can be determined alongside with a corresponding valuation $\sigma(i)$ of the cost function i to be minimized. Hence, we can define a computable function SOLVE from integer constraint systems to $\mathbb{N} \cup \{-1\}$ s.t.

$$\textsc{solve}(\phi) := \begin{cases} \sigma(i) & \text{for some } \sigma \text{ with } \sigma \models \phi \text{ if } \phi \text{ is satisfiable,} \\ -1 & \text{otherwise.} \end{cases}$$

Assume w.l.o.g. that the integer variable i we are going to minimize only takes values from $\mathbb{N}$. Then SOLVE yields an upper estimate of the cost of the optimal solution s.t. an optimal solution minimizing i can be obtained by applying a binary search scheme on the range of i as follows:

```
SEARCH(φ) :
    L := 0
    R := SOLVE(φ)
    while (L < R) do
        M := (L + R) div 2
        K := SOLVE(φ ∧ i ≥ L ∧ i ≤ M)
        if (K = −1) then L := M else R := K fi
    done
```

After termination of SEARCH, R either equals -1, indicating unsatisfiability of ϕ, or contains the optimal value of i.

4 Application specific SAT engine

4.1 Improving the optimization algorithm

Expectedly, the boolean encoding of integer constraints as described above gives rise to SAT instances of consider-

able size. The case study from [5], e.g., translates to a SAT problem with 180,000 variables. Yet, these instances could be solved by GOBLIN in relatively little time ([8]) which turned out to be due to the fact that the solution values of many integer variables occurring in the input formulae were determined solely by a relatively small number of boolean decisions, yielding a favorable propagation structure of the SAT problems entailed.

Consider e.g. the integer constraints (1): As soon as one allocation variable a_i^p of a task τ_i is assigned true by the SAT solver, the (fixed-point encoded and therefore integer-valued) worst case response time r_i of that task wrt. the current partial allocation is uniquely determined. Moreover, r_i need not necessarily be computed within the SAT solver, but can in principle be determined externally by actually carrying out the fix-point iteration (1).

This observation lead us to relocate certain, however not all integer constraints from the SAT solver to a specialized engine, called *RT-engine* in the following, leaving the SAT checker with a much smaller propositional formula ϕ to be solved.

Algorithm 1 shows the integration of the RT-engine with the standard DPLL procedure [12] which is usually employed in modern SAT checkers as GOBLIN[3]. The resulting

Algorithm 1 Modified DPLL procedure

RTSAT(ϕ, σ)
  explanation := DEDUCE(ϕ, σ)
  if *explanation* $\neq \varnothing$ **then**
    $\phi := \phi \wedge \left(\bigvee_{k \in \textit{explanation}} \neg k\right)$
    return
  explanation := RT_ENGINE(σ)
  if *explanation* $\neq \varnothing$ **then**
    $\phi := \phi \wedge \left(\bigvee_{k \in \textit{explanation}} \neg k\right)$
    return
  if no free variables left **then exit_with**(σ)
  b := SELECT_UNASSIGNED_VARIABLE(ϕ)
  v := ONE_OF {`true`, `false`}
  RTSAT($\phi, \sigma \cup \{b \mapsto v\}$)
  RTSAT($\phi, \sigma \cup \{b \mapsto \neg v\}$)

procedure, referred to as RTSAT, aims at finding a satisfying valuation σ for ϕ by successively extending partial Boolean valuation until either a conflict is encountered, forcing the algorithm to backtrack, or all variables have been assigned without a conflict, in which case a satisfying valuation is found. To extend a partial valuation, RTSAT starts by entering a deduction phase in which all assignments that are enforced by the current valuation are executed. The deduction phase is subdivided into two parts: the first part, implemented in DEDUCE, performs deduction on the propositional level only, whereas the second part, implemented in RT_ENGINE, also checks for consequences wrt. the real-time constraints. If either part (DEDUCE or RT_ENGINE) detects a conflict, e.g. a violation of a placement constraint or a deadline, it returns a set of Boolean assignments which implies the conflict and can therefore be seen as an explanation for the conflict. This explanation, which should be as general as possible, is *learned* by the solver by conjoining a clause to ϕ which excludes that special set of assignments from future search. If, on the other hand, deduction ends without a conflict, RTSAT enters the so-called decision phase which extends the current valuation by (heuristically) selecting a free Boolean variable and exploring the consequences when assigning `true` and `false` to it.

During the incremental construction of σ the SAT solver in particular assigns (and unassigns) the allocation variables a_p^i, thereby mapping tasks to ECUs, thus generating a series of partial allocations which are handed over to the RT-engine which checks them for feasibility. To this end, the RT-engine determines for each task τ_i which has already been allocated (and whose allocation variables $a_i^p,\ p \in P$ are therefore, enforced by (2), all assigned) the set

$$hp(\tau_i) := \{j \mid \forall p \in P : \sigma(a_j^p) = \sigma(a_i^p) \ \wedge \ \xi(\tau_j) > \xi(\tau_i)\}$$

of tasks with higher priorities that have (in the current partial allocation) been mapped onto the same ECU. With these sets on hand, the RT-engine, whose implementation is sketched in algorithm 2, calculates the worst case response time of each placed task τ_i according to equation(1). In case of a deadline violation the RT-engine returns the allocation variables of those tasks whose placement on the same ECU causes the violation and whose valuation therefore can serve as an explanation for the conflict.

Algorithm 2 Implementation of the RT-engine

RT_ENGINE(σ)
  for all $i \in \{j \mid \exists p \in P : a_j^p = \sigma(\texttt{true})\}$ **do**
    $r_i^{n+1} := \_\text{LO}(c_i) + \sum_{j \in hp(\tau_i)} \_\text{LO}(c_j)$
    do
      $r_i^n := r_i^{n+1}$
      $r_i^{n+1} := \_\text{LO}(c_i) + \sum_{j \in hp(\tau_i)} \left\lceil \frac{r_i^n}{\_\text{HI}(t_j)} \right\rceil \_\text{LO}(c_j)$
    until $(r_i^n = r_i^{n+1}) \vee (r_i^{n+1} > d_i)$
    if $r_i^{n+1} > d_i$ **then**
      return $\{a_j^p \mid \exists p \in P \exists j : \sigma(a_i^p) = \sigma(a_j^p) = \texttt{true}\}$
  for all *communications* **do**
    ...
  return $\varnothing$

As some of the variables involved in the fix-point computation performed by algorithm 2 might be determined

[3] For sake of clarity we have opted to present the DPLL procedure in a recursive notation. Actually, GOBLIN implements DPLL iteratively, using backjumping instead of chronological backtracking.

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

by the SAT solver, RT_ENGINE has to deal with the fact that the bit representation of those variables may contain Boolean variables which are yet unassigned. In order to avoid inappropriate pruning of the search tree, the feasibility check within the RT-engine has to compute a *lower bound* on the worst case response time in this case. This is achieved by "completing" the bit patterns of partially assigned integer variables by application of the functions _LO or _HI, respectively, which yield the smallest / biggest integer value admissible under the given partial assignment. Hence, RT_ENGINE only returns a violation of constraints if future assignments of variables occuring in the bitwise representation of the integer variables involved in that conflict would also result in a conflict.

4.2 Residual SAT problem

The fact that some of the integer constraints occuring in ϕ can be solved by both, the SAT solver *and* the RT-engine, leaves us with some freedom of choosing where to place these constraints. If *all* such constraints are handled by the RT-engine, which thus contains the whole schedulability analysis, the remaining SAT part of the problem is rather compact. We demonstrate this using the following simple example:

Let $\mathcal{A} = (\{p_0, p_1\}, \mu^{\mathcal{A}}, \{p_0, p_1\})$ be an architecture consisting of two ECUs, some memory amount on each ECU and one connecting bus system, and $\mathcal{T} = \{\tau_0, \tau_1, \tau_2, \tau_3\}$ a task network of four tasks with the following constraints: $\pi_0 = \pi_1 = \pi_3 = \{p_0, p_1\}$, $\pi_2 = \{p_1\}$, $\delta_0 = \{\tau_2\}, \delta_1 = \emptyset$, $\delta_2 = \{\tau_0\}$, $\gamma_0 = \{(\tau_1, s_{01}, d_{01}), (\tau_2, s_{02}, d_{02})\}$, $\gamma_1 = \gamma_3 = \emptyset$ and $\gamma_2 = \{(\tau_3, s_{23}, d_{23})\}$, and some memory consumption $\mu_i^{\mathcal{T}}$ for all tasks. For this example, the residual SAT problem is depicted in example 1. Herein m_i is the total amount of memory consumed by the tasks allocated on ECU i.

Assume now that a partial assignment σ with $\sigma(a_0) = 0, \sigma(a_1) = 0, \sigma(a_2) = 1, \sigma(a_3) = 0$ yields a deadline violation which is detected by the RT-engine. In this case RT_ENGINE returns the explanation $\{a_0^0, a_1^0, a_2^1, a_3^0\}$ which is used by RTSAT to build the conflict clause $(\neg a_0^0 \vee \neg a_1^0 \vee \neg a_2^1 \vee \neg a_3^0)$ that is added to the problem and that will keep the solver from investigating the same configuration again.

4.3 Exploiting symmetries

Sometimes conflict clauses turn out to be less general than they could be. Consider, e.g., a conflict due to a deadline violation on the bus system which is detected by the RT-engine.

Usually, such a conflict will not only occur in the current configuration but also in all symmetric configurations where the same set of messages has to be transmitted via the connecting bus. In this situation we would like to learn a conflict clause which captures all those isomorphic conflicts.

To this end, we introduce a special Boolean variable k_j^i for each message sent by task τ_i to τ_j which indicates whether the message is sent via the communication bus or not. The new variables allow to formulate conflict clauses which abstract from a specific placement of tasks in the desired way.

Currently, it is ongoing work to find more such symmetries we can exploit to speed up the search.

Example 1 SAT problem for given problem instance

$a_0^0, a_0^1, a_1^0, a_1^1, a_2^0, a_2^1, a_3^0, a_3^1$: **boole**;
a_0, a_1, a_2, a_3 : **integer range** 0 **to** 1;
m_0, m_1: **integer range** 0 **to** N;

$a_0^0 + a_0^1 = 1$;
$a_1^0 + a_1^1 = 1$;
$a_2^0 + a_2^1 = 1$;
$a_3^0 + a_3^1 = 1$;
$(a_0^0 \rightarrow a_0 = 0) \wedge (a_0^1 \rightarrow a_0 = 1)$;
$(a_1^0 \rightarrow a_1 = 0) \wedge (a_1^1 \rightarrow a_1 = 1)$;
$(a_2^0 \rightarrow a_2 = 0) \wedge (a_2^1 \rightarrow a_2 = 1)$;
$(a_3^0 \rightarrow a_3 = 0) \wedge (a_3^1 \rightarrow a_3 = 1)$;
$\neg a_2^0$;
$a_0 \neq a_2$;
$m_0 = 0 +$ **if** (a_0^0) **then** $\mu_0^{\mathcal{T}}(0)$ **else** 0 **fi**
+ **if** (a_1^0) **then** $\mu_1^{\mathcal{T}}(0)$ **else** 0 **fi**
... ;
$m_1 = 0 +$ ... ;
$m_0 \leq \mu^{\mathcal{A}}(0) \wedge m_1 \leq \mu^{\mathcal{A}}(1)$;

4.4 Re-use of learned clauses

In addition to learning of conflict clauses *within* the RT-SAT procedure, we also hand on learned clauses (which exclude infeasible configurations) to successive calls of RT-SAT within the binary search. By this means, future runs of RTSAT are prevented from following paths to partial assignments that have been proven infeasible in previous runs.

This re-use of derived clauses necessitates, however, some bookkeeping concering the origin of those clauses. In fact we have to distinguish two different sources of infeasible configuration: Firstly, general infeasible configurations, for example deadline violations due to overload situations, and secondly infeasible configuration that depend on the actual refinement of the optimization clause, for example exceeding the utilization limit on a bus where this limit is part of the optimization goal. Clauses from the first class can be re-used in each invocation of the binary search, whereas clauses belonging to the second class depend on the cur-

rent refinement of the optimization clause. Therefore, we store conflict clauses which depend on optimization goals in a conflict repository and construct for each invocation of RTSAT within the binary search a set of learned facts which are sound wrt. the current optimization refinement.

5 Experimental results

5.1 Comparison with other approaches

Our first preliminary evaluation was done on the well known example from [5], which we also evaluated in [8] using a pure SAT approach (see section 3). The example consists of 43 tasks sending 36 messages, some memory consumption for each task, 8 ECUs with different amount of memory, one central token-ring bus connecting all ECUs, and placement restrictions for some of the tasks. The result is given in table 1.

Table 1. Optimization results for [5] using simulated annealing (SA), SAT checking, RTSAT and RTSAT with learning (RTSAT-L)

	SA	SAT	RTSAT	RTSAT-L
TRT	8.67ms	8.55ms	8.55ms	8.55ms
run-time	n.a.	48min	105sec	50sec

The optimization criterion was chosen in a way such that the load on the central token-ring bus is minimized, represented by the token rotation time (TRT). All SAT-based approaches yield the optimal solution of this problem, but the run-time for the RTSAT approach is improved by a speedup of nearly 50. Note, that due to the specific construction of the RT-engine we can very easily change bus systems and scheduling paradigms, if schedulability analysis for these systems exist. For instance, replacing the token-ring by a CAN bus in the above example instance leads to nearly the same runtime. In comparison to the pure SAT approach this results in a speedup of about 400.

Our approach can be applied to many more optimization problems in the domain of real-time systems where optimality of the solution is wanted. As an example we refer to the problem reported by Burns et al. in [13], where the optimization task is to minimize the utilization of a periodic server, which is used for a set of sporadic tasks, in the presence of other concurrent tasks under a fixed-priority preemptive scheduling. The results in [13] show, that due to harmonic relation between periods of server and sporadic tasks there exist one optimal solution which is an isolated outlier within the solution space, without having good neighborhood solutions. Therefore, it is nearly impossible to find this solution using neighborhood-based optimization strategies like simulated annealing, tabu search or genetic algorithms. In fact, we modeled this problem within our framework and got the optimal solution given in [13] within a few seconds.

Another scenario where our optimal approach is beneficial is the application of multi-objective optimizations. Stochastical or heuristical approaches, like e.g. [14], produce interpolation points on a so-called Pareto front[4], but they are not able to give confidence intervals for these points. Using our technique, we are able to establish interpolation points which truly belong to the front. This is done by successively fixing one dimension of the multi-objective optimization to a set of different limits, and then searching for the optimal solution of this $n-1$-objective optimization problem for each of the fixed limits and for each dimension.

5.2 Scalability

In order to show the scalability of our RTSAT approach, we investigate numerous problem instances and scale the number of tasks as well as the number of ECUs. Figure 1 shows the runtime behavior when scaling the number of tasks, one row of data for the RTSAT approach, and the other row for the RTSAT approach with learning during binary search. We used the example from [5] as initial instance (43 tasks, 8 ECUs, one central token-ring bus, partly restricted placement). The task set is then raised from 43 tasks (communicating 36 messages) of the original instance to 100 tasks (communicating 58 messages) by adding new task graphs, but without new memory constraints[5]. The optimization goal was to minimize the bus load, which can easily be achieved by minimizing the load of non-local communications:

$$\min_{\forall \Pi : \mathcal{T} \to P} \left\{ \sum_{m_{i,j}=(\tau_j,l,\cdot)\in\gamma_i : \Pi(\tau_i)\neq\Pi(\tau_j)} \frac{c_{m_{i,j}}}{t_i} \right\},$$

where $m_{i,j}$ is the message sent over the bus, and $c_{m_{i,j}}$ is the load of the message on the bus (derived by their size l, frame packaging and overhead in case of using the CAN bus, speed of bus, etc.).

From figure 1 the benefit of using learning techniques becomes apparent. However, all solutions lead to feasible configurations and the runtimes of the optimization approach is quite low (at maximum 32 minutes for RTSAT-L). The number of boolean variables to be assigned by the SAT solver ranges from 19,000 for the smallest benchmark to 48,000 for the largest task system.

[4] A solution is part of the Pareto-front if an further improvement of one of the multi-objective parameters leads to worsening all other parameters. The Pareto-front builds the surface of a $n-1$ dimensional corpus of feasible solutions in a n-objective optimization.

[5] This is because memory is the crucial factor in the instance from [5]

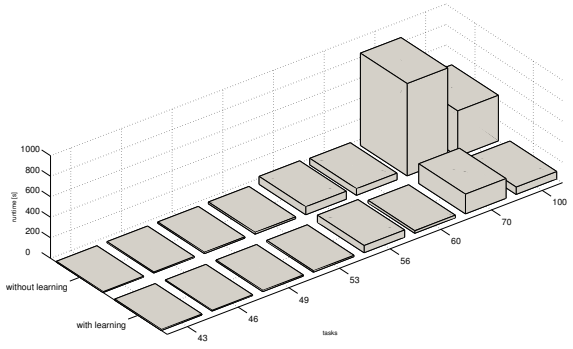

Figure 1. Runtimes of optimizing an increasing set of tasks with and without learning.

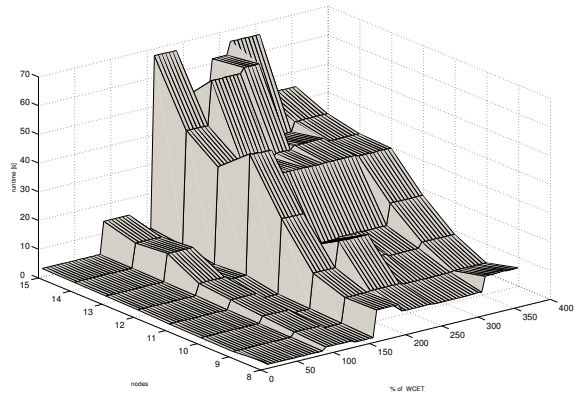

Figure 2. Scalability I: Increasing WCETs vs. increasing number of ECUs.

Before we discuss the scalability with respect to the number of ECUs, let us first make a remark regarding the different instances: In general it turns out that runtime increase is not only sensitive to the size of instances (in terms of number of tasks and number of ECUs), but also to the tightness of the instances. This is the reason for needing much more time on the original instance from [5] in comparison to an instance with less restrictions (in terms of memory consumption, placement restrictions and real-time concerns). In order to capture scalability considering tightness we perform some hundred (synthetic) benchmarks, based on the example given in [5]. The results of these investigations are shown in figure 2 to figure 4.

Figure 2 shows the runtime behavior considering an increasing number of ECUs and increasing real-time concerns in terms of worst case execution times. The benchmark is based on [5] but without memory constraints. The number of ECUs is increased from 8 to 15, and the WCET of all tasks ranges over 20% to 400% of the WCET given in [5]. Again, optimization objective is minimizing the bus load.

Figure 2 shows the dependability of the runtime from the number of ECUs which can clearly be traced back to the quadratic increase of the number of boolean variables (see section 4.2 for coding details). However, the real-time concerns have a major impact as well. This can be seen from the steep edge at WCETs of 160%: Up to this value the optimization procedure was able to produce a solution with constant minimal bus utilization due to mostly local communication. For WCETs that are higher than 160% the scheduling for a large set of tasks was not feasible only using local communications due to overhead situations on single ECUs. In general, finding a more distributed allocation for tasks is more complex, because more alternatives have to be traversed. At this point the increasing complexity induced by more ECUs clearly becomes visible. Further fluctuations in the runtime can be traced back to runtime variations SAT checker inherently have due to their heuristic implementation for exploring search spaces (which theoretically has exponential size).

The results of varying the bus speed instead of the number of ECUs is shown in figure 3. Again, we use the example from [5] without memory constraints and with a central bus architecture connecting 8 ECUs. Objective function is minimizing the bus load under increasing WCETs and increasing bus speed.

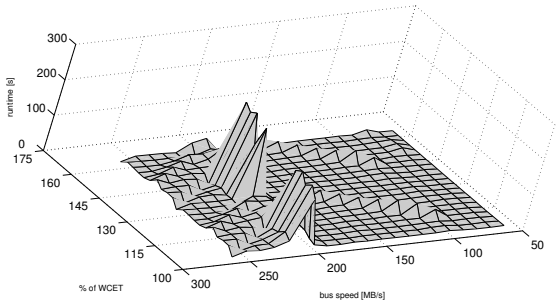

Figure 3. Scalability II: Increasing WCETs vs. increasing speed of bus.

The results are more or less independent from both parameters, even for unsatisfiable or satisfiable instances: Up to 80MB/s the instances are in general unsatisfiable due to the transmission of messages enforced by restricted placements in [5], after this border they are partly unsatisfiable, depending on the values of the tasks WCET. For the waves and the sparely distributed peaks in the surface of measure-

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE

ment we have at the moment no general explanation beside the natural fluctuations of SAT checkers mentioned above.

Finally, we try to more and more tighten the instance. Therefore we use the well known example from above, keep the memory consumption and remove the restricted placement. Then we range the memory of all 8 ECUs from 125% down to 88% and concurrently introduce randomly generated placement restrictions from no restrictions for allocation decisions to 70% of all allocation decisions are forbidden. The results can be viewed in figure 4, where optimization goal again was to minimize the bus load.

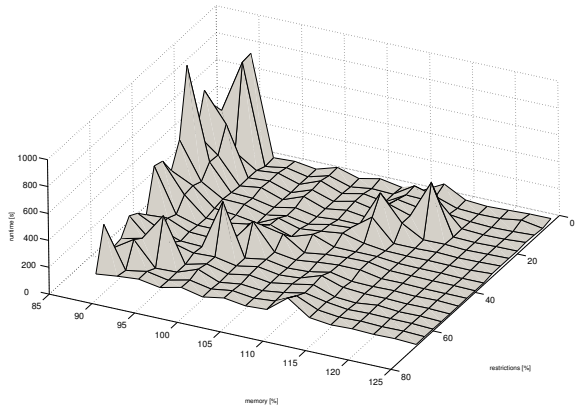

Figure 4. Scalability III: Increasing memory vs. increasing number of restrictions.

Obviously, the restrictions have no major impact, but the memory limitation has. Note, that instances with restrictions of more than 60% are in general unsatisfiable, which is not visible in the runtimes for them. Less memory on all nodes leads to much fewer valid solutions. Thus, the optimization procedure has to inspect a lot of configurations which leads to a deep traversing in the decision tree and forces a lot of expensive backtracking within the SAT solver. This effect is amplified by less restrictions due to the higher number of allowed configurations, which can be seen in the increasing runtimes in the left upper corner of figure 4.

By further reducing the memory amount on each ECU downto to 40% of the original instance, we observed a satisfiability front at about 83% memory for which at the same time the most runtime for finding the optimal solution was needed (not shown in figure 4). Beyond this front the runtimes of the SAT checker were highly seceding towards a few seconds for each instance. Obviously, the latter effect is caused by the ability of early pruning parts of the search space. For reaching the satisfiability border this effect cannot be exploited, hence in future work we will investigate more general conflict clauses for these specific situations, as we have already established for communication overload (see section 4.2).

5.3 Different application scenarios

Finally, we will sketch the usability of the approach presented in this paper to application scenarios, which are typical for the state space exploration of embedded real-time systems design. Hence, we will give an intuition how our method can be used to reach the claims of our introduction.

Firstly, one of the main problem, e.g., the automotive industry has to overcome is the nearly irrepressible increase of the number of networked ECUs. The approach presented in this paper could be one of the key technologies to break the old paradigm of equating one function to one ECU. The main goal of reducing the number of ECUs is achievable due to solving the task allocation problem and at the same time optimizing for the minimal number of ECUs. We can implement this optimization objective within our framework simply by introducing additional boolean variables $\texttt{ecu}_i$ for all ECUs in the SAT part of our model. Whenever a task τ_j is allocated on an ECU i, a value of TRUE is assigned to the boolean variable $\texttt{ecu}_i$:

$$\texttt{ecu}_i = \bigvee_{\tau_j \in \mathcal{T}} \texttt{a}_j^i$$

The optimization function can now be formulated by

$$\sum_{\forall p \in P} \texttt{ecu}_p \leq L,$$

where L can be refined during binary search. We applied this to the example from [5] with a CAN bus instead of a token-ring and yield a result of 6 ECUs and a bus load of 40.1% on the CAN bus (Only optimizing bus utilization yields a bus load of about 30% and uses 8 ECUs).

Secondly, due to supplier-dominant developing of complex embedded systems in automotive as well as avionics industry there is a need for late changes of initially tailored design spaces, caused by requirement violations of supplier's parts of the system. Handling such scenarios must carefully consider other parts of the system, such that the amount of change is preferably small. To this end there is a need for an incremental integration, where the optimization process will start with some pre-allocations and changed parameters or additional new tasks. Similarly to the problem above we are able to incorporate such change-sensitive optimization by simply introducing new allocation stores within the SAT part which represent the configuration of the pre-allocation (this could be constants). In comparison with the current partial allocation variables the SAT checker is able to sum the number of changes. Again, we can give a limit L on the number of changes within the optimization process which then is decreased during binary search, leading to the

minimal number of changes for a given system with respect to a pre-allocation.

Furthermore, preserving slackness for each task in order to master the occurrence of larger WCETs than expected can help to avoid such incremental changes. This can be supported in our approach by two ways: Firstly, the optimization is driven to an uniform utilization of all ECUs. In [8] we have shown the applicability using the pure SAT approach. Secondly, we can remove the fixed values of WCETs from the RT-engine and put it under control of the SAT part, as described in section 4.2. Intervals, e.g. $\pm 20\%$, limit the magnitude of each WCET change:

$$\subset_i : \textbf{integer range } (c_i \cdot 0.8) \textbf{ to } (c_i \cdot 1.2);$$

where c_i is the estimated WCET for task τ_i and $\subset_i$ is the integer variable used in the SAT part of the optimization procedure. By this means it is possible to optimize towards a maximal expansion of the WCETs, e.g. $\sum \subset_i > L$, where L is increased during binary search.

In a nutshell the above described extensions towards more complex handling of development process driven optimizations can be achieved by simply adding new constraints to the SAT part of the RTSAT solver, which means that the RT-engine can remain unchanged. Nevertheless, for performance reason we think also about synthezising the RT-engine tailored for a given problem instance.

6 Conclusion

We have presented an enhanced SAT-based approach which tailered to solve the task allocation problem for distributed real-time systems. In contrast to heuristic allocation approaches, which are known to behave well only if an application specific parametrisation of the algorithm can be found, our experiments show that we can achieve optimal assignments for industrial-sized applications from scratch, supporting accurate time predictions. This is enabled by a special SAT solver extension that uses a real-time engine for the calculation of response times for tasks and messages.

Further work is planned in two directions: On the one hand we plan to improve the optimization engine by investigating new constraint tightenings and generalisation rules, we will analyze the impact of problem specific choices of variables during the decision phase, and we will evaluate which part of the problem model should remain in the SAT part and which should be moved into the RT-engine. On the other hand we will enhance the class of application scenarios by enabling the use of hierarchical topologies of embedded networks and by incorporating different scheduling analysis, like EDF, TTOsek, etc.

In the future, this new approach will allow the application of optimal design space exploration for embedded real-time applications with various complex system properties, like bursts, loops, mode changes etc. which are well studied from the schedulability point of view, and which can easily be integrated into the RT-engine. Mastering such systems will be one key technology in upcoming embedded systems development processes.

References

[1] J. Coffman. *Computer and Job-Shop Scheduling Theory*. John Wiley, 1976.

[2] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2), 1995.

[3] K. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems*. PhD thesis, University of York, 1994.

[4] P. Altenbernd. *Timing Analysis, Scheduling, and Allocation of Periodic Hard Real-Time Tasks*. PhD thesis, Universität Paderborn, 1996.

[5] K. Tindell, A. Burns, and A. Wellings. Allocating hard real time tasks. In *Real Time Systems Journal*, volume 4(2), 1992.

[6] P. Pop, P. Eles, Z. Peng, and V. Izosimov. Schedulability-driven partitioning and mapping for multi-cluster real-time systems. In *Proc. of the Euromicro Conference on Real-Time Systems*, 2004.

[7] T. Blickle, J. Teich, and L. Thiele. System-level synthesis using evolutionary algorithms. In *Proc. of the Design Automation for Embedded Systems*, 1998.

[8] A. Metzner, M. Fränzle, C. Herde, and I. Stierand. Scheduling Distributed Real-Time Systems by Satisfiability Checking. In *Proc. of the Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE Computer Society, 2005.

[9] M. Fränzle and C. Herde. Efficient SAT engines for concise logics. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2850 of *LNAI*. Springer, 2003.

[10] A. Bender. Design of an optimal loosely coupled heterogeneous multiprocessor system. In *Proc. of the European conference on Design and Test*. IEEE Computer Society, 1996.

[11] D. Peng, K. Shin, and T. Abdelzaher. Assignment and scheduling communicating periodic tasks in distributed real-time systems. *Software Engineering*, 23(12), 1997.

[12] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *Communications of the ACM*, 5:394–397, 1962.

[13] R. Davis and A. Burns. Hierarchical Fixed Priority Pre-emptive Scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE Computer Society, 2005.

[14] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA — a platform and programming language independent interface for search algorithms. In *Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science, pages 494 – 508. Springer, 2003.

Proceedings of the 27th IEEE International
Real-Time Systems Symposium (RTSS'06)
0-7695-2761-2/06 $20.00 © 2006 IEEE